Lecture 2

Functions & Modules

Labs Start this Week

- Labs are Wednesday in Upson 142
  - Can do them at home (instructions on web)
  - Bring a laptop if you come to the lab
- Lab is due **before beginning** of next lab
  - Can check off *on the day it is handed out*
  - Can check off during *consultant hours*
- **Cannot spend lab time on previous lab**

Announcements

If Not Done Already

- Enroll in Piazza
- Sign into CMS
  - Fill out the Survey
  - Complete AI Quiz
- Do AI Quiz by **Friday**!

Reading (Optional)

- Readings this week
  - Chapter 1 (browse)
  - Chapter 2 (in detail)
  - Chapter 3 (in detail)
- Next week
  - Sections 8.1, 8.2, 8.4, 8.5

[xkcd.com]

Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,…)

- **Examples** (math functions that work in Python):
  - `round(2.34)`
  - `max(a+3,24)`

Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,…)

function name

argument

- **Examples** (math functions that work in Python):
  - `round(2.34)`
  - `max(a+3,24)`

Arguments can be any **expression**

Built-In Functions

- You have seen many functions already
  - Type casting functions: `int()`, `float()`, `bool()`
  - Dynamically type an expression: `type()`
  - Help function: `help()`
  - Quit function: `quit()`

Arguments go in (), but `name()` refers to function in general

- One of the most import functions is `print()`
  - `print(exp)` displays value of `exp` on screen
  - Will see later why this is important

Built-in Functions vs Modules

- The number of built-in functions is small
  - http://docs.python.org/3/library/functions.html
- Missing a lot of functions you would expect
  - **Example**: `cos()`, `sqrt()`
- **Module**: file that contains Python code
  - A way for Python to provide optional functions
  - To access a module, the `import` command
  - Access the functions using module as a *prefix*

Example: Module math

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

Example: Module math

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Example: Module math

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Functions require math prefix!

Example: Module math

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Functions require math prefix!

Module has variables too!

Example: Module math

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Functions require math prefix!

Module has variables too!

Other Modules

- io
  - Read/write from files
- random
  - Generate random numbers
  - Can pick any distribution
- string
  - Useful string functions
- sys
  - Information about your OS

Reading the Python Documentation

Python Software Foundation docs.python.org/3/library/math.html

Gaming News Research Commentary #comments Financial Travel Shopping Technical TroubleShooting Developer BBQ Misc

9.2. math — Mathematical functions — Python 3.6.2 documentation

Python » English 3.6.2 Documentation » The Python Standard Library » 9. Numeric and Mathematical Modules » Quick search Go | previous | next | modules | index

Table Of Contents

Previous topic

9.1. numbers — Numeric abstract base classes

Next topic

9.3. cmath — Mathematical functions for complex numbers

This Page

Report a Bug
Show Source

9.2. math — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. Number-theoretic and representation functions

math.**ceil**(*x*)
Return the ceiling of *x*, the smallest integer greater than or equal to *x*. If *x* is not a float, delegates to `x.__ceil__()`, which should return an `Integral` value.

math.**copysign**(*x*, *y*)
Return a float with the magnitude (absolute value) of *x* but the sign of *y*. On platforms that support signed zeros, `copysign(1.0, -0.0)` returns *-1.0*.

math.**fabs**(*x*)
Return the absolute value of *x*.

math.**factorial**(*x*)
Return *x* factorial. Raises `ValueError` if *x* is not integral or is negative.

math.**floor**(*x*)
Return the floor of *x*, the largest integer less than or equal to *x*. If *x* is not a float, delegates to `x.__floor__()`, which should return an `Integral` value.

math.**fmod**(*x*, *y*)
Return `fmod(x, y)`, as defined by the platform C library. No
C standard is that `fmod(x, y)` be exactly (mathematically; to infinite precision) equal to `x - n*y` for some integer *n* such that the result has the same sign as *x* and magnitude less than `abs(y)`. Python's `x % y` returns a result with the sign of *y* instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be

http://docs.python.org/library

Reading the Python Documentation

Python Software Foundation docs.python.org/3/library/math.html

9.2. math — Mathematical functions — Python 3.6.2 documentation

Python » English 3.6.2 Documentation » The Python Standard Library » 9. Numeric and Mathematical Modules » Quick search Go | previous | next | modules | index

Table Of Contents

9.2. math — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

math.ceil(*x*)

Return the ceiling of *x* as a float, the smallest integer value greater than or equal to *x*.

This Page

Report a Bug
Show Source

-1.0.

math.**fabs**(*x*)
Return the absolute value of *x*.

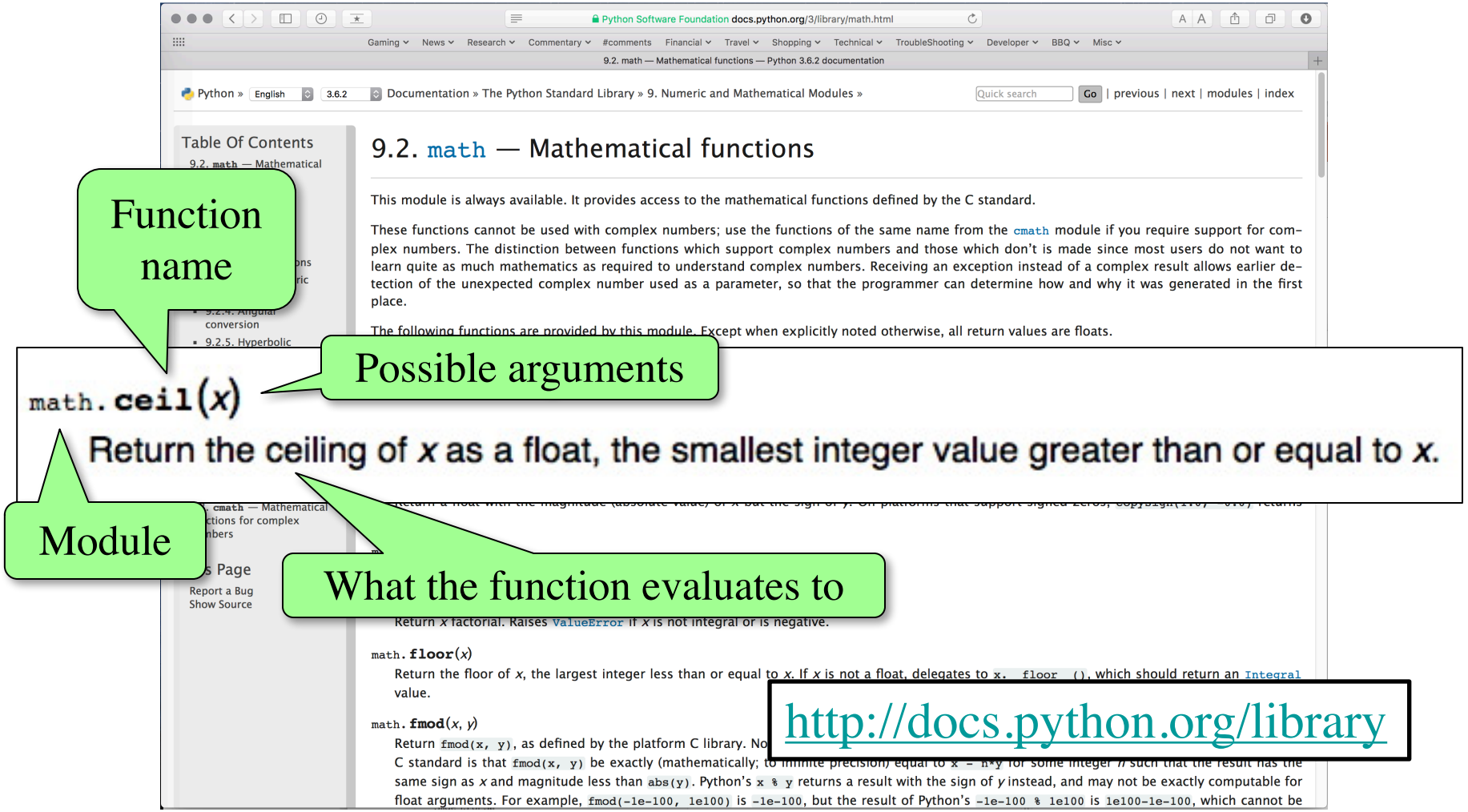
math.**factorial**(*x*)
Return *x* factorial. Raises ValueError if *x* is not integral or is negative.

math.**floor**(*x*)
Return the floor of *x*, the largest integer less than or equal to *x*. If *x* is not a float, delegates to x.__floor__(), which should return an Integral value.

math.**fmod**(*x*, *y*)
Return fmod(x, y), as defined by the platform C library. No
C standard is that fmod(x, y) be exactly (mathematically; to infinite precision) equal to x - n*y for some integer n such that the result has the same sign as *x* and magnitude less than abs(y). Python's x % y returns a result with the sign of *y* instead, and may not be exactly computable for float arguments. For example, fmod(-1e-100, 1e100) is -1e-100, but the result of Python's -1e-100 % 1e100 is 1e100-1e-100, which cannot be

http://docs.python.org/library

Reading the Python Documentation



Interactive Shell vs. Modules

```
[wmwhite@dhcp-hol-172]:~ > python
Python 3.6.1 |Anaconda 4.4.0 (x86_64)| (default, May
 11 2017, 13:04:09)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57
)] on darwin
Type "help", "copyright", "credits" or "license" for
 more information.
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>>
```

```
"""
A simple module.

This file shows how modules work

Author: Walker M. White (wmw2)
Date:   August 25, 2017 (Python 3 Version)
"""

x = 1+2      # I am a comment
x = 3*x
x
```

- Launch in command line
- Type each line separately
- Python executes as you type

- **Write in a text editor**
  - We use Komodo Edit
  - But anything will work
- Load module with import

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

Single line comment
(not executed)

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

Single line comment
(not executed)

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

Single line comment
(not executed)

Commands
Executed on import

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

Single line comment
(not executed)

Commands
Executed on import

Not a command.
import **ignores this**

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

Python Shell

```
>>> import module
>>> x
```

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

"**Module data**" must be prefixed by module name

Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
```

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

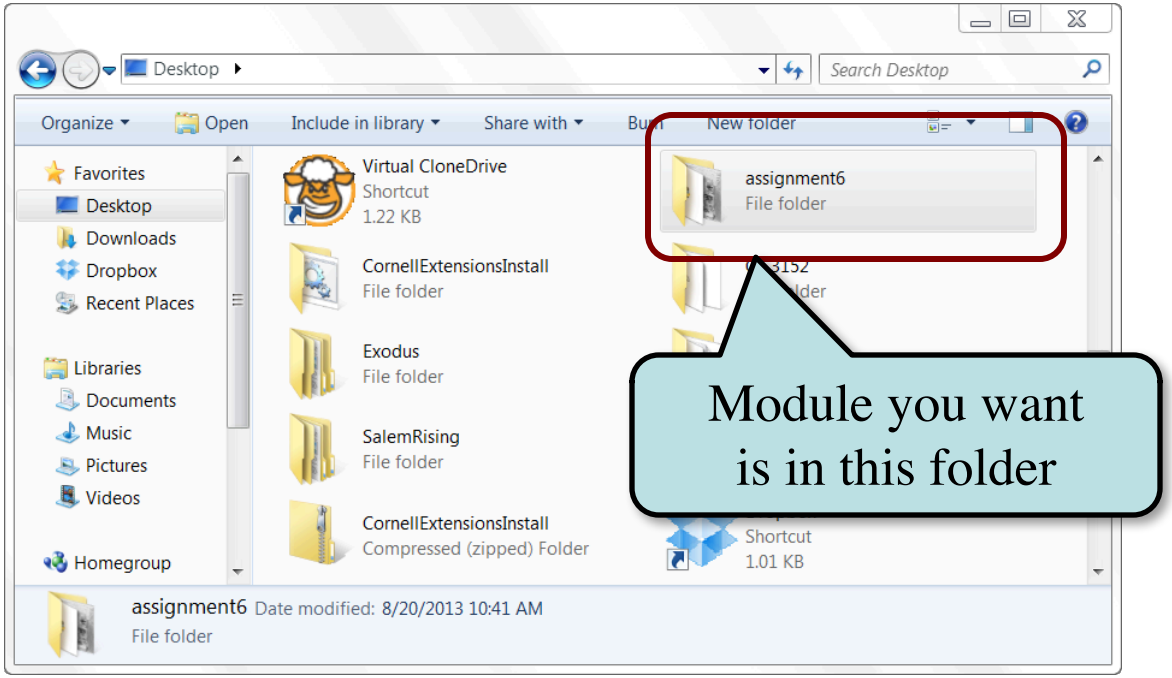
"**Module data**" must be prefixed by module name

Prints **docstring** and module contents

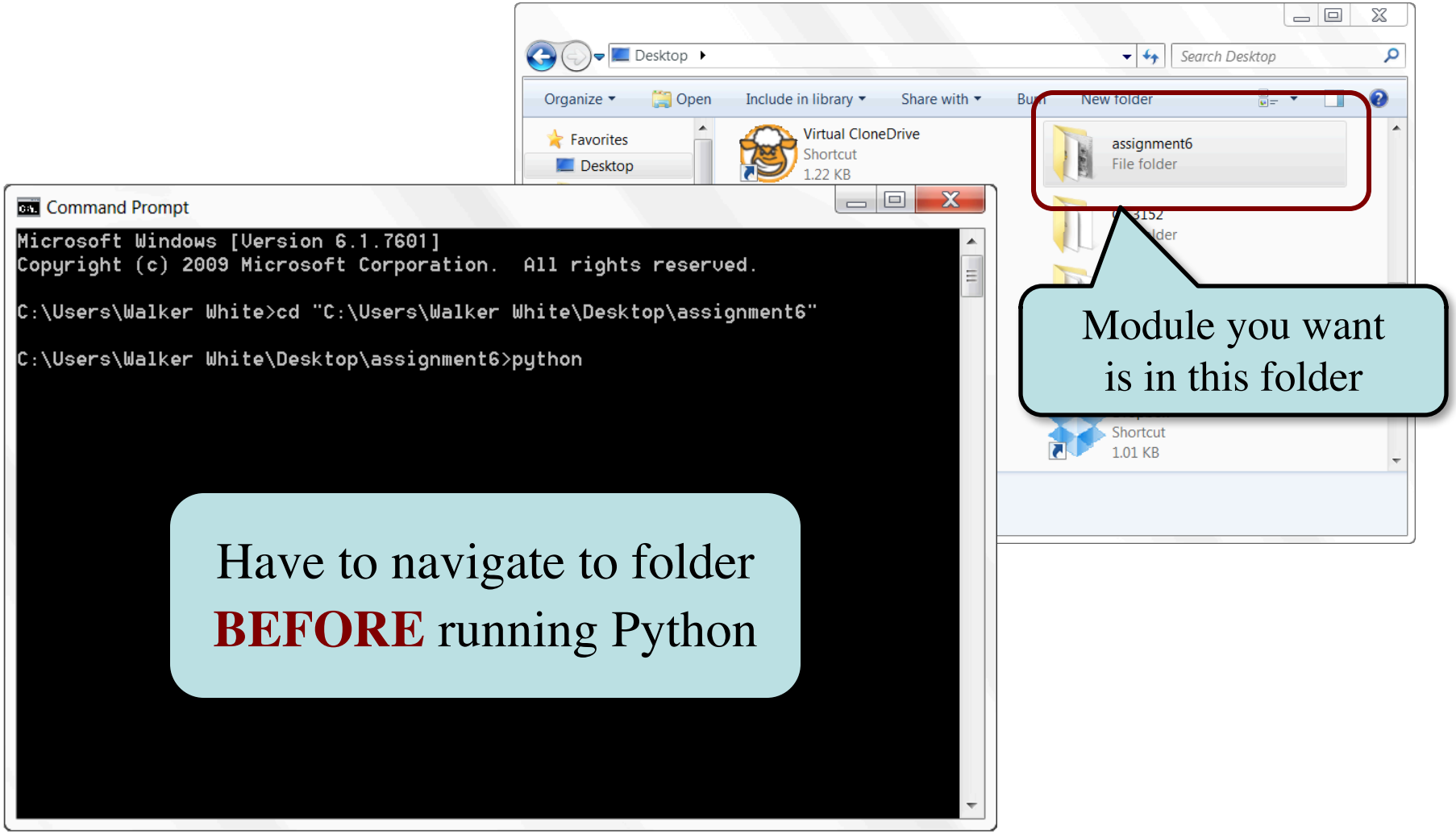
Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
>>> help(module)
```

Modules Must be in Working Directory!



Modules Must be in Working Directory!



Modules vs. Scripts

Module

- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> import temp
>>> temp.to_fahrenheit(100)
212.0
>>>
```

Script

- Behaves like an application
  - **Example**: helloApp.py
- Run it from command line:

```
python helloApp.y
```

Modules vs. Scripts

Module

- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> import temp
>>> temp.to_fahrenheit(100)
212.0
>>>
```

Script

- Behaves like an application
  - **Example**: helloApp.py
- Run it from command line:

```
python helloApp.y
```

Files look the same. Difference is how you use them.

Scripts and Print Statements

module.py

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

script.py

```
""" A simple script.

This file shows why we use print
"""


# This is a comment
x = 1+2
x = 3*x
print(x)
```

Scripts and Print Statements

module.py

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

script.py

```
""" A simple script.

This file shows why we use print
"""

# This is a comment
x = 1+2
x = 3*x
print(x)
```

Only difference

Scripts and Print Statements

module.py

```
modules — -bash — 62×24
[wmwhite@Ryleh]:modules > python module.py
[wmwhite@Ryleh]:modules >
```

- Looks like nothing happens
- Python did the following:
  - Executed the assignments
  - Skipped the last line
    (‘`x`’ is not a statement)

script.py

```
modules — -bash — 62×24
[wmwhite@Ryleh]:modules > python script.py
9
[wmwhite@Ryleh]:modules >
```

- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line
    (Prints the contents of `x`)

Scripts and Print Statements

module.py

```
modules — -bash — 62×24
[wmwhite@Ryleh]:modules > python module.py
[wmwhite@Ryleh]:modules >
```

- Looks l
- Python
  - Execu assignments
  - Skipped the last line
    (‘`x`’ is not a statement)

script.py

```
modules — -bash — 62×24
[wmwhite@Ryleh]:modules > python script.py
9
[wmwhite@Ryleh]:modules >
```

his time!

the following:

- Executed the assignments
- Executed the last line
  (Prints the contents of `x`)

When you run a script,
only statements are executed

Next Time: Defining Functions

Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

modules — python — 52×20

```
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
>>>
```

Function Definition

- Command to **do** the function
- Belongs inside a module

Go to Anything

plusone.py

... Lecture 3 > modules > plusone.py > Ln: 6 Col: 4 UTF-8 Python

```
1  """
2  A module with a function definition
3
4  Author: Walker M. White (wmw2)
5  Date:   August 25, 2017 (Python 3 Version)
6  """
7
8  def plus(n):
9      """
10     Returns: the value of n+1
11     """
12     return (n+1)
13
```

Next Time: Defining Functions

Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

modules — python — 52×20

```
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
>>>
```

Can **call** as many times as you want

Function Definition

- Command to **do** the function
- Belongs inside a module

Go to Anything

plusone.py ×

... Lecture 3 > modules > plusone.py > Ln: 6 Col: 4 UTF-8 Python

```
1  """
2  A modu
3
4  Author
5  Date:
6  """
7
8  def plus(n):
9      """
10     Returns: the value of n+1
11     """
12     return (n+1)
13
```

But only define function **ONCE**

Next Time: Defining Functions

Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

modules — python — 52×20

```
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
>>>
```

arguments inside ()

Can **call** as many times as you want

Function Definition

- Command to **do** the function
- Belongs inside a module

Go to Anything

plusone.py

... Lecture 3 > modules > plusone.py > Ln: 6 Col: 4 UTF-8 Python

```
1  """
2  A modu
3
4  Author
5  Date:
6  """
7
8  def plus(n):
9      """
10     Returns: the value of n+1
11     """
12     return (n+1)
13
```

But only define function **ONCE**

Functions and Modules

- Purpose of modules is **function definitions**
  - Function definitions are written in module file
  - Import the module to call the functions
- Your Python workflow (right now) is

1. Write a function in a module (a .py file)
2. Open up the Terminal/Command Prompt
3. Move to the directory with this file
4. Start Python (type `python`)
5. Import the module
6. Try out the function